Grant Agreement No.: 823783
Call: H2020-FETPROACT-2018-2020

Topic: H2020-FETPROACT-2018-01
Type of action: RIA

D5.1
WeNet's Diversity-Aware Interactions I

Revision: v.0.1

Work package	WP5
Task	T5.1, T5.2, T5.3, T5.4
Due date	30/04/2020
Submission date	30/04/2020
Deliverable lead	CSIC
Version	0.1
Authors	Nardine Osman (CSIC), Carles Sierra (CSIC), Ronald Chenu-Abente (UNITN), Qiang Shen (UNITN), Fausto Giunchiglia (UNITN), Bruno Rosell (CSIC), Athina Georgara (CSIC), Juan Antonio Rodriguez (CSIC), Thiago Freitas (CSIC), Marco Schorlemmer (CSIC)
Reviewers	Loizos Michael (OUC), Isidoros Perikos (OUC)

Abstract	WeNet's main objective is achieving a diversity-aware, machine-mediated paradigm for social interactions. This deliverable focuses on WeNet's diversity-aware interaction model that enables our online interactions while

	ensuring they are privacy-compliant, diversity-aware, and in more general terms, fulfil our ethical requirements. The deliverable is divided into two main parts. The first focuses on the issue of mediating our online interactions in such a way that ensures certain properties are met. For this, a normative-based decentralised architecture is proposed. The second supports WeNet's main objectives in helping out people by connecting them together, which is achieved by finding the right group of people for a given task. For this, a grouping mechanism is proposed.
Keywords	team formation, interaction model, norms, diversity, privacy, values

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	28/02/2020	1st version to be reviewed internally	Nardine Osman (CSIC)
V1.0	30/04/2020	Final version submitted	Nardine Osman (CSIC)

Disclaimer

Copyright notice

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:*	**R**	
Dissemination Level		
PU	Public, fully open, e.g. web	✔
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to the WeNet project and Commission Services	

** R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

Co-funded by the Horizon 2020 Framework Programme of the European Union

EXECUTIVE SUMMARY

WeNet's main objective is achieving a diversity-aware, machine-mediated paradigm for social interactions. This deliverable focuses on WeNet's diversity-aware interaction model that enables our online interactions while ensuring they are privacy-compliant, diversity-aware, and in more general terms, fulfil our ethical requirements.

The deliverable is divided into two main parts. The first focuses on the issue of mediating our online interactions in such a way that ensures certain properties are met (such as privacy). For this, we propose an architecture for the WeNet's interaction model that is based on normative systems. While normative systems have excelled at addressing issues such as coordination and cooperation, they have left a number of open challenges. The first is how to reconcile individual goals with community goals, without breaching the individual's privacy. The evolution of norms driven by individuals' behaviour or argumentation have helped take the individual into consideration. But what about individual norms that one is not willing to share with others? Then there are the ethical considerations that may arise from our interactions, such as, how do we deal with stereotypes, biases, or racism, or how to avoid the abuse of community resources. Our proposal accounts for individual needs while respecting privacy and adhering to the community's ethical code. We propose a decentralised architecture for normative systems that, along with the community norms, introduces individual's requirements to help mediate the interaction between members.

The second part of the deliverable focuses on finding the right group of people for a given task. One of WeNet's main objectives is helping out people by connecting them together. For example, if one needs help preparing for an exam, then WeNet is responsible for finding which group of students is best to work with, taking into consideration students' diverse backgrounds. Finding the right and effective group is not an easy task, and the importance of diversity in groups is well documented. This second line of work is concerned with developing a diversity-driven grouping mechanism that allocates student teams to tasks, initially focusing on diversity in competences. We note, however, that how to group people cannot be conducted in a generic way, independent of the goal of the team. As such, our team formation algorithms will be strongly influenced by the WeNet pilot scenarios and their defined goals, which are yet to be finalised. So while this deliverable proposes the scenario of allocating student teams to tasks, we note that as soon as the pilot scenarios are agreed upon, the plan will be to revisit our proposed algorithm to take into consideration the particular requirements of the agreed upon pilot scenarios.

Co-funded by the Horizon 2020
Framework Programme of the European Union

TABLE OF CONTENTS

Co-funded by the Horizon 2020 Framework Programme of the European Union

Chapter 1

Open Social Systems

This chapter proposes an architecture for the WeNet's interaction model that is based on normative systems. While normative systems have excelled at addressing issues such as coordination and cooperation, they have left a number of open challenges. The first is how to reconcile individual goals with community goals, without breaching the individual's privacy. The evolution of norms driven by individuals' behaviour or argumentation have helped take the individual into consideration. But what about individual norms that one is not willing to share with others? Then there are the ethical considerations that may arise from our interactions, such as, how do we deal with stereotypes, biases, or racism, or how to avoid the abuse of community resources. This chapter is concerned with accounting for individual needs while respecting privacy and adhering to the community's ethical code. We propose a decentralised architecture for normative systems that, along with the community norms, introduces individual's requirements to help mediate the interaction between members.

1.1 Introduction

Normative systems have attracted a lot of attention in the multi agent systems community as one approach to maintain the autonomy of agents while ensuring community goals and aspirations are fulfilled. Norms essentially specify the rules of interaction: what one can (or cannot) do, when, under what conditions, etc. Normative systems copy how human societies function, and they can be compared to social norms that govern society's behaviour or organisational norms that mediate interactions in organisations [9].

While normative systems have excelled at addressing issues such as coordination and cooperation [2], they have left a number of open challenges. The first is how to reconcile individual goals with community goals, without breaching the individual's privacy. A number of approaches have been studied to take the individual into consideration, such as norm synthesis techniques that would help norms evolve based on individuals' behaviour [7], or norm evolution that would allow the individuals to reason about norms through argumentation [8]. But what about individual norms that one is not willing to share with their fellow community member? For example, imagine a community norm that states that a donation cannot be below 5€ and an individual norm that states that a donation cannot exceed 50€. Another open challenge are the ethical considerations that may arise from our interactions, such as, how do we deal with stereotypes, biases, or racism, or how to avoid the abuse of community resources, to name a few.

In other words, the question this chapter addresses is how can we make sure that an individual will have their needs taken into consideration while we ensure their privacy is respected and the community's ethical code is not violated. To address these issues, this chapter proposes a decentralised architecture for normative systems that, along with the community norms, introduces individual's requirements to help mediate the interaction between members. Section 1.2 presents our proposal in brief, Section 1.3 introduces the notation used in this chapter, Section 1.4 introduces the decentralised architecture addressing the challenges discussed above, while Section 1.5 provides a motivating example, before concluding with Section 1.7.

Co-funded by the Horizon 2020
Framework Programme of the European Union

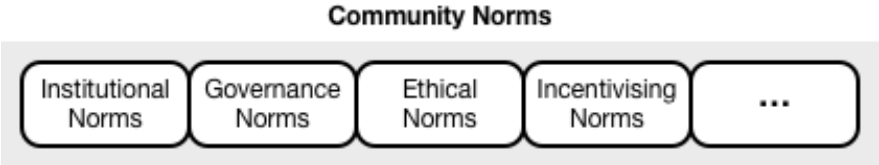


Figure 1.1: Community norms

1.2 Proposal

To address the issues presented above, we first say that in addition to community norms, there are also individual norms that describe the individual's rules of interaction with others.

Norms, as illustrated earlier, specify what actions are acceptable for that specific individual, who can the individual interaction with, and under what circumstances. While normative systems have focused a lot on the action, 'what' can one do, we highlight in this chapter the other crucial aspect of interactions: 'who' can one interact with. The 'who' aspect has been implicit until now, usually hidden under the 'what' action specification. In an increasingly hyperconnected world, we choose to make the 'who' more explicit in our proposal. To achieve this, we require users to have profiles describing them, such as describing their gender, their age, their relationships, etc. With such profiles, rules on who to interact with can then be specified. For example, one individual norm can then say 'only seek the support of female friends during my breakup period', while another can say 'never ask my ex-husband for help'. As such, and in addition to community norms and individual norms, the individual profile becomes another crucial element for mediating our interactions.

Both the individual's norms and profile may be divided into a private and shared part. In what follows, we present the norms and the profiles in more detail.

1.2.1 Norms

As per the above we distinguish between community norms and individual norms.

- **Community norms**. These norms are the community's agreed upon norms. Any action (represented by a message exchange) in the peer-to-peer network must be coherent with them. We consider an action acceptable by the community when it doesn't violate any of the community's norms.

  We note community norms can be categorised into a number of groups (Figure 1.1). For example, institutional norms can describe the rules of behaviour in the given community (following the concept of electronic institutions [4]). Governance norms can describe the rules of who has the right to change existing norms and how. Ethical norms can describe what is considered ethical and what actions are deemed unethical, and hence, unacceptable in the community. Incentivising norms can help provide incentives for community members to behave in a certain way, such as encouraging benevolent behaviour, say to help maintaining the community and fulfilling its objectives.

  One can even imagine re-using, adapting, or building on top of existing norms. For example, a new social network may re-use the institutional norms of an existing social network and adapt them to their community's particular needs.

- **Individual norms**. These norms represent particular aspects of the relationship of the human with her machine and with the community. For instance, a prohibition to pop-up a message during a siesta unless coming from a relative. Or one can filter messages coming from people that they do not deem trustworthy. As most individual norms are private, some 'unethical' behaviour may be codified at this level and remain unnoticed, such as a private norm requiring to never show messages coming from old men.

  In general, individual norms may implement certain behaviour that may not be fully aligned with the community values and norms. In cases of conflict between community norms and individual private

Co-funded by the Horizon 2020 Framework Programme of the European Union

ones, community norms prevail concerning actions within the community. For example, if community norms prohibit discriminating against women, then an action like excluding females from a given activity will be prohibited. However, individual private norms prevail when concerning actions local to one's machine. For instance, while community norms may prohibit discriminating against women, one's private norms can enforce requests coming from women to be suppressed (ignored).

We note that individual norms can further be divided into two parts: private norms and shared norms.

- **Private norms** are norms that are private and are never shared with other machines (e.g. 'never show messages coming from old men'). Their impact is restricted as other machines do not have access to these norms.
- **Shared norms** are norms that travel with messages so that other people's machines can take them into consideration (e.g. when specifying 'do not ask the help of people outside Barcelona', the receiving machine can check the location of its human, even if this data is private as this data never leaves the machine and is not shared with others).

1.2.2 Profiles

Generally speaking we assume we have two types of profiles that we can intuitively describe as follows.

- **Private profile.** This is the set of features that are private to (and hence, accessible only by) the human's own machine. For instance, if `gender("A",female)` is part of Alice's private profile this means that Alice's machine has permission to use Alice's gender in the reasoning.
- **Shared profile**. This is a set of features that can be shared with (or made accessible to) others, both the humans and their machines. There are several approaches, both centralised and decentralised, that one can choose from for making information public. However, in this proposal, we suggest sharing the public profile by communicating it to other machines on an as-needed basis.

Of course, humans decide what part of their profile is public and what part is kept private.

The notion of private profile is quite intuitive. We want to keep private what we do not want the others to know. This issue of privacy has always been around but it has become of paramount importance with the pervasive use of the Web and the Social Media. In the past we were protected for free by our space and time limitations: it would take some time to go from place A to place B and this time would increase with distance. The phone lifted some time barriers, but the propagation of information would still be limited by the fact that we were able to choose who to interact with and, in any case, the communication would only happen in pairs. Television lifted other barriers, allowing for zero time one-to-many communication, but still information was very much verified and under control and in many cases regulated by law. The Social Media have lifted the last barrier: now everybody can talk with everybody and say whatever they prefer with basically no limitations (with the first limitations being established by the most recent legislation, for instance, GDPR in Europe).

The social media have made it possible to replicate and hugely expand what has always been the case in the real world. Now anybody can share information with anybody, virtually the entire world population, in zero time and no space constraints. This motivates the focus on privacy and hence the need for a private profile.

But this is only part of the story. First of all, *the notion of privacy is not an absolute notion*. There is information that I may be willing to share with my family but not with my friends and even less with my enemies. For example people are usually very happy to share information about the location of their children in a certain moment of time, for instance the fact that they go to a school with a certain address and that lectures will end at 1pm, with a person with a car that maybe has a child who goes to the same school. But they would never be willing to share this information with a person they do not fully trust. In social relations, the notion of *privacy is fully contextual* in the sense that it depends on the current situation and also in the objectives that one wants to achieve.

The contextuality, and therefore non-absoluteness, of privacy brings up the key observation which underlies the need for both a public and a private profile. To provide another example which integrates the one about the child who needs to be picked up from school, suppose I have a certain medical condition, e.g.,

Co-funded by the Horizon 2020 Framework Programme of the European Union

diabetes. This is sensitive information, namely information with many more constraints for its circulation. In general, most people would not talk about their disease, but, for instance, a person with diabetes, if too low in her level of sugar in the blood, would be very happy to let others know about this. And not only of the need for sugar but also of the fact that the reason is diabetes, as this would increase the urgency of the intervention. In social relations there is always *a tension between privacy and transparency*. In almost any interaction with other people we trade-off some privacy (about us, about our family, friends, ..., anybody) as a key enabler for obtaining information, support, information from others.

The notion of public profile captures exactly this need of transparency, meaning by this the sharing information as key to enabling social interactions. Clearly, the public profile is *contextual*, where the person we interact with is a crucial component of the relevant context, and mostly *dynamic*. There is in fact very little information, if any, that we are willing to always share with others; maybe our name, but also in this case it is easy to think of exceptions. Furthermore the public profile, like the private profile, will change in time because of multiple reasons, e.g., change of one's job or of the place where one lives. The contextuality and dynamicity of the public profile will require its continuous update and revision (issues addressed by the WeNet platform). This consists of a process which will be enforced by the local peer, as driven by its user, and which will consist of performing a set of abstraction operations [6] on the private profile.

1.3 Notation

We first present, in this Section, the notation used in the remainder of this chapter. We say let CN describe the set of community norms, PrR and ShR describe the sets of private and shared individual norms, respectively, and PrP and ShP describe the private and share profiles, respectively. We view a profile as a set of features. To specify which agent does a set of norms or profile describe, we use the sub index of that agent. For example, PrR_A describe's A's private norms whereas ShP_B describes B's shared profile.

We say a profile is a set of features, and we specify features as propositions. For example, we say *g*ender("A",female) to state that A's gender is female and *l*oc("A",barcelona) to state that A's location is in Barcelona. As for norms, we specify these as "if then" statements that talk about actions (similar to the rule-based norms of [5]), and we use the deontic operators O and F to describe obligations and prohibitions, accordingly. For example, *F*(display("A",M)) states that it is forbidden to display the message M to A.

1.4 Architecture and associated operational model

In Figure 1.2, the schema of the peer-to-peer architecture for our proposed normative system is presented. Each user has a machine, that may run all or some of its computations on a remote server (depending on the complexity of the norms and their computational requirements). Each user interacts with its machine through a user interface.

As illustrated earlier, each user specifies their profile and individual norms. The profile is divided into private (PrP) and shared (ShP) parts, and the norms into private (PrR) and Shared (ShR) parts.

The norm engine at each machine will have both a reactive and proactive behaviour.

- **Reactive Behaviour.** This allows the norm engine to react to messages received (usually representing the actions being performed), and there are two types of messages that a machine can receive:
  - *A message from the user interface.* When a user performs an action, it is translated into a message that is sent to the machine through the user interface. The message includes the shared norms and a copy of the sender's shared profile. Upon the receipt of such a message, the norm engine needs to first verify that the message does not violate any of the norms, this includes the community norms and the sender's individual norms (both private and shared). A conflict resolution mechanism should address any conflicting norms that may arise. If the action violates any of those norms, an error message is sent back to the user. However, if the action obeys the norms, then the norm engine needs to decide what to do next, usually translated into sending messages to other peers. This decision follows from the community and individual norms (both private and shared), and takes the user's profile (both public and shared) into account as needed.

Co-funded by the Horizon 2020 Framework Programme of the European Union

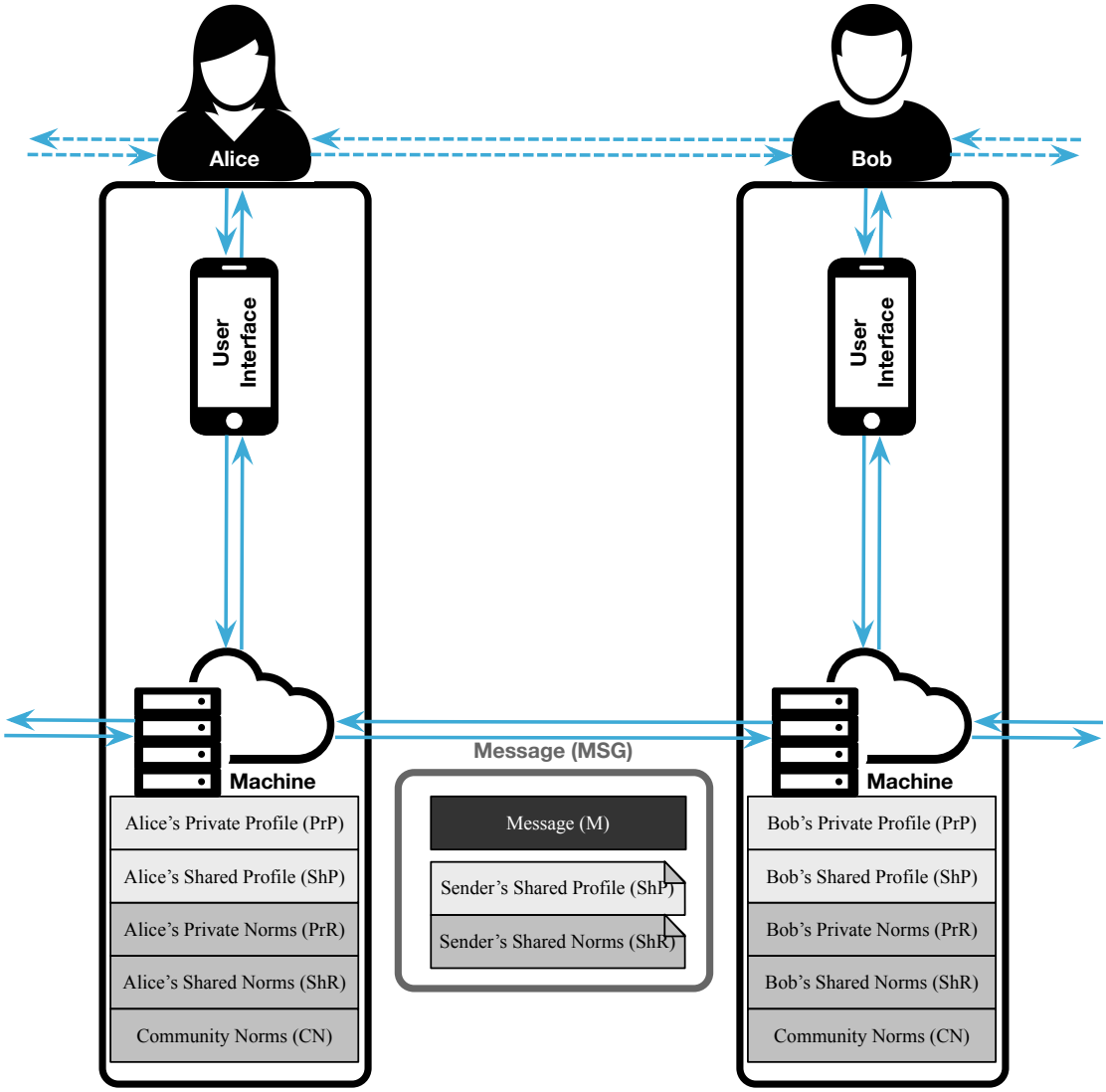


Figure 1.2: Basic (distributed) architecture

– *A message from another machine.* As in the previous case, the norm engine needs to first verify that the message does not violate any of the community norms. This re-checking upon receipt ensures that the sender's norm engine has not been manipulated to cheat. If the message violates any of the community norms, then it may either be discarded, or if the community norms require sanctioning, then the appropriate sanctions should be executed.[1]

  However, if the action obeys the community norms, then the norm engine needs to decide what to do next, which is usually translated into sending messages to other peers and/or sending messages to the user interface. This decision takes into consideration the community norms, the norms attached to the message, and the individual private and shared norms. This ensures that the machine abides with its human's private norms without leaking any of their private norms and profile.

- **Proactive Behaviour.** This allows the norm engine to proactively perform actions as required by the norms. In other words, the norm engine does not simply react to actions performed by users, by can

[1]Note that in this document, we do not get into the details of where are norm engines physically located (locally or not), as this is an issue to be decided on the implementation level (which is outside the scope of this paper). Nor do we get into the discussion of how to ensure the security of the norm engine and prevent its manipulation. But it is worth noting that traditional approaches of distributed systems can be investigated here to address manipulation, such as following the blockchain approach.

Co-funded by the Horizon 2020
Framework Programme of the European Union

proactively take an action itself, and as required by the norms. For example, incentivising norms might remind a user to complete their profile, if this has been neglected for some time, or remind the user of how much their contribution to their community is valued, if they haven't been active lately. To be proactive, a machine will require access to the community norms and individual private norms, as well as its human's private and public profile.

1.5 Motivating Example

In this example, imagine having four people involved: Alice (A), Bob (B), Carol (C), and Dave (D). Say Bob, Carol and Dave are on Alice's contact list, and Bob is on Carol's contact list. The community norms (CN) specify how a help request is propagated in the social network. They state that every time a machine receives a help request, it needs to decide whether it displays it to its user (Lines 32–35, Figure 1.3), and whether it needs to forward it and to whom (Lines 24–31, Figure 1.3). We note that the person making the request will decide the maximum number of hops accepted when looking for volunteers (`Hops`) and the minimum trustworthiness required (`Trust`). As these are specified for a given task, they are sent along with the help request message (Line 38, Figure 1.3).

As for individual norms, imagine Carol has a private norm that says ignore help requests from females (i.e. do not display such requests, as show in Lines 13–17, Figure 1.3). Alice, on the other hand, has a private norm and a shared one. The private one specifies that only those who are close by (in Barcelona) get to see her help requests (Lines 10–12, Figure 1.3). The shared one specifies that none of her requests may be displayed to Bob (Lines 19–22, Figure 1.3). As Bob is her ex-husband, she prefers that Bob does not see her requests, though she is happy for his machine to still receive her requests as she is interested in using his social network. Hence, she only prohibits the display of the message to Bob.

Now concerning people's profiles, some information such as gender, location, or trust in others may be kept private (Lines 1–4, Figure 1.3), or made public (Lines 6–8, Figure 1.3). For example, Alice's private profile specifies her trust in her contacts: in this case, 'low' for Bob and 'high' for Carol and Dave. Similarly, Carol's private profile specifies her trust in her contact Bob as 'high' and her current location as being in London. Bob, Dave and Alice are happy to make share their gender and location with others through their shared profiles.

Now given these profiles and norms, imagine that Alice is running late at work and she needs someone to pick up her child from school (message M). She accepts friends of friends (`Hops=2`, for connection level 2), but is looking for trustworthy volunteers only (`Trust="high"`), as illustrated in Line 38, Figure 1.3. For these norms to be enforced by other machines, Alice shares these norms along with her other shared norms and profiles by attaching them to the help request (M), resulting in the message, MSG.

As soon as the help request is sent by Alice, the norm interpreter at her machine will check whether the message complies with the community norms (CN), in which case it does. The interpreter then needs to decide what are the actions that this message entails, taking into consideration Alice's profile (private and shared), her norms (private and shared), and the community norms. According to the community norm on Lines 24–31, the interpreter decides to forward the help request to Carol and Dave, as they satisfy the requested hops and trustworthiness constraints (the trustworthiness of Bob, on the other hand, is low).

Upon receiving the message, Dave's machine now needs to check whether the message applies with the community norms, which it does. It then needs to decide what are the resulting actions of receiving this message, taking into consideration Dave's profile and norms (both private and shared), the norms attached to the message (that is, Alice's shared norms), and the community norms. According to the community norm on Lines 32–35, the request is then displayed to Dave, despite the fact that Alice forbids it in its private norm. This is because Alice's private norm is private and cannot be taken into consideration by other people's machines.

Upon receiving this message, again, Carol's machine needs to check whether the message applies with the community norms, which it does. After that, it needs to decide what are the resulting actions of receiving this message, taking into consideration Carol's profile (private and shared), her norms (private and shared), the norms attached to the message (that is, Alice's shared norms), and the community norms. In this case, and according to Carol's private norm on Lines 13–17, the help request is not displayed on Carol's mobile as it comes from a female. However, the help request is forwarded to Bob, according to the community norm

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
1   PrP_A = { trust("A","B",low) ;
2             trust("A","C",high) ;
3             trust("A","D",high) }
4   PrP_C = { trust("C","B",high) ; loc("C",london) }
5
6   ShP_A = { gender("A",female) ; loc("A",barcelona)}
7   ShP_B = { gender("B",male) ; loc("B",barcelona) }
8   ShP_D = { gender("D",male) ; loc("D", rome) }
9
10  PrR_A = { IF ¬loc(X,barcelona) ∧ requester(M)="A" THEN
11              F(display(X,M))
12            END IF }
13  PrR_C = { IF rcv_help_rqst(Sndr,"C",Trust,Hops,MSG) ∧
14               gender(Sndr)=female ∧
15               MSG=M+ShR_M+ShR_Sndr+ShP_Sndr THEN
16                   F(display("C",M))
17            END IF}
18
19  ShR_A = { IF rcv_help_rqst(Sndr,"B",Trust,Hops,MSG) ∧
20               MSG=M+ShR_M+ShR_Sndr+ShP_Sndr THEN
21                   F(display("B",M))
22            END IF }
23
24  CN = {IF rcv_help_rqst(Sndr,Rcvr,Trust,Hops,MSG) THEN
25          FOR ALL X∈Rcvr.Contacts
26            IF trust(Rcvr,X,Y) ∧
27              Y≥Trust ∧ Hops>0 THEN
28                O(snd_help_rqst(Rcvr,X,Trust,Hops-1,MSG))
29              END IF
30            END FOR
31        END IF ;
32        IF rcv_help_rqst(Sndr,Rcvr,Trust,Hops,MSG) ∧
33          type(Sndr)=machine ∧ MSG=M+ShR_M+ShR_Sndr+ShP_Sndr THEN
34            O(display(Rcvr,M))
35        END IF }
36
37  M = "Can you pick up my son from school at 5pm?"
38  ShR_M = { O(snd_help_rqst("A",machine("A"),"high",2,MSG_M)) }
```

Figure 1.3: Motivating example: profiles and norms

at Lines 24–31. Note that while Alice's trust in Bob was low, her trust in Carol is high, and Carol's trust in Bob is also high, allowing the message to be forwarded to Bob through Carol.

Upon receiving the message, Bob's machine again checks its adherence to community norms. Then, as above, it needs to decide what are the resulting actions of receiving this message, taking into consideration Bob's profile (private and shared), his norms (private and shared), the norms attached to the message (that is, Alice's shared norms), and the community norms. In this case, and according to Alice's shared norm on Lines 19–22, the help request is not displayed on Bob's mobile as Alice forbids it.

This example illustrates how our proposed system ensures the interaction between people adheres to both community norms and individual ones without jeopardizing people's privacy. It also illustrates the impact of private and shared information. For instance, private norms are better suited to control local behaviour, whereas shared norms are better suited for controlling the behaviour of other machines.

1.6 Beyond the Proposed Architecture

Our architecture for open social systems provides the foundations for the WeNet interaction model. Ongoing work that builds on this interaction model has focused (up until now) on two main issues. First, how to specify the different WeNet pilot scenarios in this architecture, and second, how to address the issue of alignment in our WeNet interactions. In what follows we present the progress of this ongoing work.

Co-funded by the Horizon 2020
Framework Programme of the European Union

1.6.1 Specifying Scenarios

To specify a WeNet scenario, say a scenario where people use the WeNet system to find fellow students who can join them for dinner, we specify the rules of interaction for that scenario. Rules of interaction may be thought of rules specifying who can do what, when, and under what condition. For example, who can organise a dinner, who can volunteer to attend, how does the system look for volunteers, and so on. These rules constitute the community norms, and as such, WeNet scenarios are specified through community norms. Naturally, additional individual norms or even task related norms can be appended, such as "I can only accept a maximum of 5 attendees at my place".

To specify community norms, a language for norms is needed. We suggest to build here on the traditional approach of using deontic logic for specifying. Deontic logic is the logic of duties, and it deals with concepts like permissions, prohibitions, and obligations, which help specify who can do what, under what conditions, and so on. In hardware systems and networks, deontic-based policy languages have been used widely in hardware systems and networks for security reasons, trust negotiation, access control, authorisation, and so on [10, 3]. In multiagent systems, several deontic based formal logics have been proposed for helping coordinate collective activities, and coordination is mainly achieved through procedural norms [1, 11].

In this project, we choose to follow the deontic approach, though we decide to simplify the syntax of the normative language by specifying norms through "`if` ... `then` ..." statements. We believe such statements are expressive enough for our scenarios (the final WeNet scenarios, currently being finalised, will either confirm this or require minor modifications to our proposed language). For example, a rule can say "If someone cancels their request then volunteers should be informed". We note, however, given the above architecture and its requirement for a decentralised approach, any change in state needs to be saved locally (e.g. if I am no longer a volunteer). This change in state is represented via the Belief-Desire-Intention (BDI) model, a model used in programming intelligent agents to help keep track of the agents' beliefs and desires and help plan their actions accordingly (through intentions).

As such, "if then" rules should be specified with this requirement in mind: they should specify the necessary changes in state. For example, the above norm can be specified as:

```
if action(cancel(R)) then believe(request(R,cancelled))
if believe(request(R,cancelled)) then msg(SV,request(R,cancelled))
```

In other words, if the user performs the action of cancelling their request (`action(cancel(R))`), then locally, the user updates the status of the request accordingly (`believe(request(R,cancelled))`), and this results in informing the selected volunteer of this change (`msg(SV,request(R,cancelled))`).

An initial proposal for the norm language is presented below. The syntax essentially states that norms are composed of a condition part and a consequence part. Both parts can be complex, constructed by using conjunctions, disjunctions, and negations of conditions/consequences. A simple condition can then either represent a past event or action (e.g. a deadline is reached, the user presses the cancel button, the user receives a message asking if they are interested in an activity), or a change in the BDI state (e.g. user now beliefs an activity is cancelled). Similarly, simple consequences can represent an action to be performed (e.g. inform the selected volunteer of my cancellation), or a BDI state change to be applied (e.g. I now belief the task is cancelled).

```
Norm            :: if Condition then Consequence
Condition       :: Condition and Condition |
                   Condition or Condition |
                   not Condition |
                   Simple_Condition
Consequence     :: Consequence and Consequence |
                   Consequence or Consequence |
                   not Consequence |
                   SimpleConsequence
```

Figure 1.4: Norm syntax

Co-funded by the Horizon 2020 Framework Programme of the European Union

While the WeNet pilot scenarios are getting finalised, a simple scenario where people can use WeNet to make requests and volunteer is currently being implemented with the above syntax as an exercise to test the expressiveness of the language.

1.6.2 Interaction Alignment

Given that interactions are mediated via norms, this line of work focuses on the issue of norm alignment due to misunderstandings. For example, say we have a scenario where people can look for volunteers to join them in some activity (such as running, football, or dinner). And say there is a community norm that states that users must be "punctual" when attending an activity. The norm is specified as: $\texttt{If chosen\_volunteer}(\texttt{X}, \texttt{Activity})\ \texttt{Then obliged}(\texttt{be\_punctual}(\texttt{X}, \texttt{Activity}))$, which essentially states that if a user $\texttt{X}$ is the chosen volunteer for a given activity ($\texttt{Activity}$), then $\texttt{X}$ must be punctual for that activity. However, understanding punctuality is very subjective. One person might accept their volunteer to be 10 minutes late, while another might believe this is rude behaviour and rate the volunteer poorly. Problems will essentially arise when there is an agreement that many users are not being punctual. Of course, there may be different reasons for people not abiding by the punctuality norm. For example, the majority may be students with part-time jobs who are usually tight on time and hence are frequently running late. But in some cases, a norm is not adhered to due to our diverse understanding of the norm. So what is very late for one person might be considered being on time for another. When problems are detected due to such misunderstandings —namely, people are not abiding by the punctuality norm due to a diverse understanding of "punctuality"— then this implies that the norm needs to be modified to address the problem at hand. For example, the norm can be changed to explicitly note that 15 minutes late is not acceptable in this community. This is essentially clarifying the semantics for punctuality for community users. For instance, the new norm can state that: $\texttt{If chosen\_volunteer}(\texttt{X}, \texttt{Activity}) \wedge \texttt{activity\_time}(\texttt{Activity}, \texttt{T}) \wedge \texttt{activity\_location}(\texttt{Activity}, \texttt{L})$ $\texttt{Then obliged}(\texttt{arrive\_before}(\texttt{X}, \texttt{L}, \texttt{T} + 5))$. This essentially states that the user $\texttt{X}$ *must* arrive to the location of the activity $\texttt{L}$ at most 5 minutes late ($\texttt{T} + 5$), where $\texttt{T}$ is the time of the activity.

In summary, detecting problems that may arise due to our diverse understandings and addressing these problems by adapting the norms as an approach to help aligning our understanding is the objective of this ongoing work. But how do we detect that people are not abiding by the norms and a change is needed? Two different approaches may be applied here: either depend on the users to explicitly flag lateness, or have the system automatically detect late arrivals (say by using geo-location). Although the more urgent question is how to learn which norm might be best to apply. Here, we have three different approached to consider. First, we may consider Reinforcement Learning (RL). Starting with a pre-defined set of rules, we could apply RL to simulate interactions and learn what the expected behaviour is in such situations. Here, we may also consider providing arguments to improve the RL process. Besides, the arguments could also be used to explain why certain decision was taken, thus dealing with the "black-box" problem of machine learning approaches. Second, Inverse Reinforcement Learning (IRL) may be considered. This approach uses the same idea of using interactions to learn, but in this case the system observes real interactions and then infers a reward function. Third, Supervised Learning with Arguments may also be considered. Using supervised learning, one could use the arguments to support the rules that the system is learning. In this case, all learned rules must be explained by the given arguments. To learn, the system receives the examples plus the arguments that explain those examples. We are currently in the process of assessing the pros and cons of the different approaches in order to make an informed decision on how to move forward.

1.7 Conclusion

This chapter has proposed a decentralised architecture for normative systems that introduces individual norms, while ensuring the privacy of people. One aspect that has been overlooked in this chapter and left for future work is the conflict resolution mechanism. Having people specify their own norms will probably result in conflicting rules, and a mechanism will be needed to address such conflicts.

Our ongoing work is implementing the different parts of the WeNet platform according to the architecture presented in this paper. The WeNet platform introduces different types of norms (private and shared ones) and different types of profiles (private and shared). We have also just commenced our work on implement-

Co-funded by the Horizon 2020 Framework Programme of the European Union

ing the norm engine that reacts to any action performed, ensuring the norms (both the individual and the community norms) are followed. Also, we are integrating the platform with an extended version of iLog [12] that automatically learns people's profiles from their online activity, and we note that it is the profiles that will provide the diversity information about groups of users.

As illustrated in our discussion of community norms, these norms can be used to specify the rules of interaction in a community, but also to introduce more specialised rules, such as rules specifying what is considered ethical and unethical, or rules specifying how to motivate people to act in a certain way. Future work will be experimenting with these specialised different, focusing on ethics and incentives.

Co-funded by the Horizon 2020
Framework Programme of the European Union

Bibliography

[1] Thomas Ågotnes, Wiebe van der Hoek, Juan A. Rodríguez-Aguilar, Carles Sierra, and Michael Wooldridge. On the logic of normative systems. In *Proc. of IJCAI '07*, pages 1175–1180, 2007.

[2] Giulia Andrighetto, Guido Governatori, Pablo Noriega, and Leendert W. N. van der Torre, editors. *Normative Multi-Agent Systems*, volume 4. Dagstuhl Publishing, 2013.

[3] Nicodemos C. Damianou, Arosha K. Bandara, Morris S. Sloman, and Emil C. Lupu. A survey of policy specification approaches. Technical report, Department of Computing, Imperial College of Science Technology and Medicine, London, UK, 2002.

[4] Mark d'Inverno, Michael Luck, Pablo Noriega, Juan Rodriguez-Aguilar, and Carles Sierra. Communicating open systems. *ARTIFICIAL INTELLIGENCE*, 186:38–94, 7 2012.

[5] Andrés García-Camino, Juan A. Rodríguez-Aguilar, Carles Sierra, and Wamberto Weber Vasconcelos. Constraint rule-based programming of norms for electronic institutions. *Autonomous Agents and Multi-Agent Systems*, 18(1):186–217, 2009.

[6] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial intelligence*, 57(2-3):323–389, 1992.

[7] Javier Morales, Maite López-Sánchez, and Marc Esteva. Using Experience to Generate New Regulations. In *Proc. of IJCAI '11*, pages 307–312, 2011.

[8] N Oren, Michael Luck, Simon Miles, and T Norman. *An Argumentation Inspired Heuristic for Resolving Normative Conflict*, pages 0000 – 0000. Unknown Publisher, 2008.

[9] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: off-line design. *Artif. Intell.*, 73:231–252, 1995.

[10] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[11] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multiagent systems. *Auton. Agents Multi-Agent Syst.*, 11(3):307–360, 2005.

[12] Ilya Zeni, Mattia ad Zaihrayeu and Fausto Giunchiglia. Multi-device activity logging. In *ACM International Joint Conference on Pervasive and Ubiquituous Computing*, pages 299–302. ACM, 2014.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Chapter 2

TAIP: An Anytime Algorithm for Allocating Student Teams to Tasks

One of the main tasks of WeNet is helping out people by connecting them together. For example, if I need help preparing for an exam, then WeNet is responsible for finding which group of students is best to work with, taking into consideration students' diverse backgrounds. Finding the *right* and *effective* group is not an easy task. This chapter presents a grouping mechanism that allocates student teams to tasks. As the WeNet use cases and their relevant diversity dimensions are still under development, we commence our work on team formation by focusing on competences obtained from Fondazione Bruno Kessler. When the WeNet diversity dimensions evolve, our grouping mechanism will be adapted to the new dimensions.

In this chapter, we focus on scenarios that require teamwork, where we usually have at hand a variety of specific tasks, for which we need to form a team in order to carry out each one. Here we target the problem of matching teams with tasks within the context of education, and specifically in the context of forming teams of students and allocating them to internship programs. First we provide a formalization of the proposed team allocation problem, and show the computational hardness of solving it optimally. Thereafter, we propose *TAIP*, a heuristic algorithm that generates an initial team allocation which later on attempts to improve in an iterative process. Moreover, we conduct a systematic evaluation to show that TAIP reaches optimality, and outperforms CPLEX in terms of time.

We note, however, that team formation strongly depends on the goal of the team. How to group people cannot be conducted in a generic way, independent of the goal of the team. As such, and because the exact WeNet pilot scenarios are still under development, this documents proposes the scenario of allocating student teams to tasks. Of course, the resulting algorithm is defined in such a way that it addresses the requirements of our selected scenario. As the WeNet pilots get finalised, the proposed algorithm will be revisited and adapted to the different WeNet pilots.

2.1 Introduction

In the context of education, it is increasingly common that students spend some time doing practical work in a company as part of their curriculum. This work is sometimes remunerated: companies benefit from this program as they get motivated students that will work for reduced wages, and students benefit from a first contact with the labour market. It has been found that the employability of students at the end of their studies increases thanks to these internships. Nowadays, education authorities match students with companies mostly by hand. This chapter formalises this matching process as a combinatorial optimization problem, proposes some heuristic algorithms and studies their computational complexity.

Team formation with respect to skills/expertise is a well studied topic of interest within the AI and MAS community [4]. [2] tackle the problem of team formation considering skills, communication costs, and tasks that progressively arrive in time. In the same direction, [8] propose a heuristic algorithm for forming one team of experts for a specific task. [7] propose several heuristic algorithms for forming a single robust team in order to compete a given set of tasks. The authors in [5] target the problem of partitioning a group

Co-funded by the Horizon 2020 Framework Programme of the European Union

of individuals into equal-sized teams so that each one will resolve the same task. Here we consider the problem of allocating individuals into teams of different sizes in order to resolve different tasks. In fact, our problem can be viewed as a generalization of [5].

In this work, we present and formalise an actual-world problem, the so-called *Team Allocation for Internship Programs (TAIPP)*. We characterise the complexity of the TAIPP and the search space that an algorithm that solves it must cope with. We propose how to encode the TAIPP as a linear program so that it can be solved by a general purpose LP solver. Furthermore, we propose a novel, anytime heuristic algorithm that exploits the structure of the TAIPP. As we will show, our proposed algorithm outperforms the general purpose optimizer IBM CPLEX in terms of time: it always reaches the optimal solution at least 55% faster than CPLEX, and reaches a quality of 80% in less than 20% of the time we need to construct the input for CPLEX.

As such, in what follows, in Sec 2.2 we formally describe the TAIPP, provide formal definitions of the problem's components, and study the complexity of the problem. In Sec 2.3 we provide the encoding for a linear program solver. In Sec 2.4 we propose our heuristic algorithm; while in Sec 2.5 we conduct a systematic evaluation and show the effectiveness of our algorithm.

2.2 Problem Formalization

In this section we present the individual components of the problem, discuss their intuition, and provide formal definitions. We begin with the formalization of *internship programs* and *students*, along with a thorough discussion on the essential notion of *competencies*. Then we proceed on presenting our notion of competence coverage, and show how to compute it.

2.2.1 Basic elements of the allocation problem

An *internship program* is characterised by a set of requirements on student competencies and team size constraints. For instance, think of an internship program in a computer tech company: there are 4 competence requirements *(a)* machine learning principles, *(b)* coding in python, *(c)* web development, and *(d)* fluency in Spanish language, while the required team size is 3 members; as such, for this program we need a team of three students that as a team possesses the four required competencies.

In general we can have a large variety of other constraints, such as temporal or spatial constraints, i.e., when and where the internship can be realised. However, within the scope of this chapter, we only focus on team size constraints. The required competencies are often accompanied by their level and importance. Formally, an *internship program* p is a tuple $\langle C, l, w, m\rangle$, where C is the set of required competencies, $l : C \to \mathbb{R}^+ \cup \{0\}$ is a required competence level function, $w : C \to (0, 1]$ is a function that weighs the importance of competences, and $m \in \mathbb{N}_+$ is the team size required. The set of all internship programs is denoted with P, with $|P| = M$.

A *student* is characterised by their competencies, and their competence levels. Formally, a student s is represented as a tuple $\langle C, l\rangle$, where C is the set of already acquired competencies, and $l : C \to \mathbb{R}_+ \cup \{0\}$ is a competence level function, and hence $l(c)$ is the student's competence level for competence c. The set of all students is denoted with S, with $|S| = N$. Given $p \in P$, we denote the set of all size-compliant teams for p as $\mathcal{K}_p = \{K \subseteq S : |K| = m_p\}$.[1]

But size is not enough, we need that the members of a team are suitable for the *competencies* requested by a company. We assume that there is a predefined ontology that provides a fixed and finite set of competencies C along with relations among them. We further assume that the ontology is a tree graph, where children denote more specific competencies than those of their parents. Formally, an ontology is a tuple $o = \langle C, E\rangle$ with C being the competencies-nodes and E the edges. The metric over ontologies that we will use next is the *semantic similarity*. The *semantic similarity* is given by

$$\mathrm{sim}(c_1, c_2) = \begin{cases} 1, & \text{if } l = 0 \\ e^{-\lambda l} \frac{e^{\kappa h} - e^{-\kappa h}}{e^{\kappa h} + e^{-\kappa h}}, & \text{otherwise} \end{cases} \tag{2.1}$$

[1] Note: we use the subscript s to refer to the set of competencies, competence level function, etc. of a student $s \in S$, and the subscript p to refer to the same elements of the internship's $p \in P$.

Co-funded by the Horizon 2020 Framework Programme of the European Union

where l is the shortest path in the tree between c_1 and c_2, h is the depth of the deepest competence subsuming both c_1 and c_2, and $\kappa, \lambda \in [1, 2]$ are parameters regulating the influence of l and h to the similarity metric. This is a variation of the metric introduced in [9], which guarantees the reflexive property of similarity, that is, a node is maximally similar to itself, independently of its depth. In other words, nodes at zero distance ($l = 0$) have maximum similarity. Similarly to [12], the semantic similarity between two competence lies in $[0, 1]$.

2.2.2 Computing competence coverage for students and teams

In order to evaluate how well a student fits with an internship we need some notion of coverage for each competence required by an internship by the actual competencies of a student. Thus, we define the *student coverage* of competence c by a set of competencies $A \subseteq C$ as $\mathrm{cvg}(c, A) = \max_{c' \in A}\{\mathrm{sim}(c, c')\}$.

And then, naturally, given a program p with required competencies C_p and a student s with acquired competencies C_s the competence coverage of program p by student s is:

$$\mathrm{cvg}(s, C_p) = \prod_{c \in C_p} \mathrm{cvg}(c, C_s) = \prod_{c \in C_p} \max_{c' \in C_s}\{\mathrm{sim}(c, c')\} \tag{2.2}$$

Moving now from a single student $s \in S$ to a team of students $K \subseteq S$, we need first to solve a *competence assignment problem*. That is, we need to assign to each student $s \in K$ a subset of competencies of C_p, and assume that student s is responsible for (in charge of) their assigned competencies. According to [5] we have that:

Definition 1 (*Competence Assignment Function (CAF)*)**.** *Given a program $p \in P$, and a team of students $K \subseteq S$, a competence assignment η_p^K is a function $\eta_p^K : K \to 2^{C_p}$, satisfying $C_p = \bigcup_{s \in K} \eta_p^K(s)$.*

The set of competence assignments functions for program p and team K is noted by Θ_p^K. The inverse function $\eta_p^{K\,-1} : C_p \to 2^K$ provides us with the set of students in K that are assigned to competence $c \in C_p$.

However, not all competence assignments are equally accepted. For example, consider a program p (with $C_p = \{c_1, c_2, c_3, c_4, c_5\}$), and a team $K = \{s_1, s_2, s_3\}$. An assignment η_p^K such that $\eta_p^K(s_1) = C_p$ and $\eta_p^K(s_2) = \eta_p^K(s_3) = \emptyset$ seems to be unfair—assigning all competencies as student s_1's responsibility—,while assignment $\tilde{\eta}_p^K$ such that $\tilde{\eta}_p^K(s_1) = \{c_1, c_3\}$, $\tilde{\eta}_p^K(s_2) = \{c_2, c_5\}$ and $\tilde{\eta}_p^K(s_3) = \{c_4\}$ is more fair, in terms of allocating responsibilities. In the setting of internship programs, we prefer assignments such that all students are actively participating, i.e., assignments such that $\eta_p^K(s) \neq \emptyset$ for each student s (the so-called *inclusive assignments* in [3]). At the same time, we would prefer not to 'overload' a few students with excessive responsibilities, but selecting fair competence assignments.[2] This is captured by the following definition:

Definition 2 (*Fair Competence Assignment Function (FCAF)*)**.** Given a program p, and a team of students $K \subseteq S$, a fair competence assignment η_p^K is a function $\eta_p^K : K \to 2^{C_p}$, satisfying $C_p = \bigcup_{s \in K} \eta_p^K(s)$, $1 \leq |\eta_p^K(s)| \leq \lceil \frac{|C_p|}{|K|} \rceil \; \forall s \in K$, and $1 \leq |\eta_p^{K\,-1}(c)| \leq \lfloor \frac{|K|}{|C_p|} \rfloor + 1$.

Now, given a competence assignment η_p^K, we define the *competence proximity* of a student s *wrt* a program p. To do so we take into consideration the importance of each competence and the students coverage of the assigned competencies. In the competence proximity we want to encode the following scenarios:

- the competence proximity should be as high as possible when the coverage of a competence by a student is maximum;
- the competence proximity should be as high as possible when the competence is not important;
- the competence proximity should be as low as possible when the coverage of a competence by all students is minimum.

For a competence $c \in C_p$ and a student s, we can visualise the above properties in the truth table in Table 2.1. If we think $\mathrm{cvg}(c, C_s)$ and $w_p(c)$, the importance of competence c in program p, as Boolean

[2] We remind the reader that these requirements are particular to our scenario of matching students to tasks. When the WeNet pilots' scenarios get finalised and their diversity dimensions identified, we will adapt our algorithm to any new requirements that might pop up.

Co-funded by the Horizon 2020 Framework Programme of the European Union

$\mathrm{cvg}(\downarrow) \setminus w(\rightarrow)$	0	$(0,1)$	1
0	1	1	1
$(0,1)$	1		$\sim \mathrm{cvg}$
1	1	$\sim (1-w)$	0

Table 2.1: Competence proximity truth table; cvg stands for $\mathrm{cvg}(c, C_s)$, and w for $w_p(c)$.

variables, we can interpret this table as a logical formula

$$w_p(c) \Rightarrow \mathrm{cvg}(c, C_s) \equiv \big(1 - w_p(c)\big) \vee \mathrm{cvg}(c, C_s)$$

However, $\mathrm{cvg}(c, C_s)$ and $w_p(c)$ are continuous variables in $[0, 1]$, so we model the 'or' condition of the above logical formula as the 'maximum' between the two variables. As such, we define the competence proximity of a student for an internship program as:

Definition 3 (*Student's Competence Proximity)*. Given a student $s \in S$, an internship program $p \in P$, and a competence assignment η_p, the competence proximity of s for p with respect to η_p is:

$$\mathrm{cp}(s, p, \eta_p) = \prod_{c \in \eta_p(s)} \max\big\{\big(1 - w_p(c)\big), \mathrm{cvg}(c, C_s)\big\}. \tag{2.3}$$

Moving to the competence proximity of a team of students $K \subseteq S$ for program p, we use the Nash product of the competence proximity of the individuals in K for p, with respect to some FCAF η_p. The Nash product assigns a larger value to teams where all students equally contribute to their program, rather than to teams where some students have a small contribution.

Definition 4 (*Team's Competence Proximity)*. Give a team K a program $p \in P$, and a competence assignment η_p, the competence proximity of team K for program p is:

$$\mathrm{cp}(K, p, \eta_p^K) = \prod_{s \in K} \mathrm{cp(s, p, \eta_p^K)}. \tag{2.4}$$

For a team K and a program p its competence proximity varies depending on the competence assignment at hand. We define the best competence assignment as the *fair* one (Definition 2) that maximizes the competence proximity:

$$\begin{aligned}
\eta_p^{K\,*} &= \underset{\eta_p^K \in \Theta_p^K}{\arg\max} \{\mathrm{cp}\big(K, p, \eta_p^K\big)\} \\
&= \underset{\eta_p^K \in \Theta_p^K}{\arg\max} \prod_{s \in K} \mathrm{cp(s, p, \eta_p^K)} \\
&= \underset{\eta_p^K \in \Theta_p^K}{\arg\max} \prod_{c \in \eta_p(s)} \max\big\{\big(1 - w_p(c)\big), \mathrm{cvg}(c, C_s)\big\}
\end{aligned}$$

Finding the *best* competence assignment is optimization problem itself. Even though the above is not a linear optimization problem, it can be easily linearized by considering the logarithm of $\mathrm{cp}(\,\cdot\,)$:

$$\begin{aligned}
\eta_p^{K\,*} &= \underset{\eta_p^K \in \Theta_p^K}{\arg\max} \{\mathrm{cp}\big(K, p, \eta_p^K\big)\} \equiv \underset{\eta_p^K \in \Theta_p^K}{\arg\max} \{\log\{\mathrm{cp}\big(K, p, \eta_p^K\big)\}\} \\
&= \underset{\eta_p^K \in \Theta_p^K}{\arg\max} \log\Big\{ \prod_{s \in K} \mathrm{cp(s, p, \eta_p^K)}\Big\} = \underset{\eta_p^K \in \Theta_p^K}{\arg\max} \sum_{s \in K} \log\big\{\mathrm{cp(s, p, \eta_p^K)}\big\}
\end{aligned}$$

2.2.3 The team allocation problem as an optimisation problem

Finding a good allocation of students to a collection of internship programs is yet another optimization problem that tries to maximize the *overall* competence proximity of all teams for their assigned internship

Co-funded by the Horizon 2020 Framework Programme of the European Union

program. That is, for a single program p, the best candidate team is the one that maximizes the competence proximity: $K^* = \arg\max_{K \in \mathcal{K}_p} \text{cp}(K, p)$. K^* is the best candidate when a single program is at hand. For a set of programs P, with $|P| > 1$, we need to maximize the competence proximity of all candidate teams with their corresponding programs. Suppose we have a team assignment function $g : P \to 2^S$, which maps each $p \in P$ with a team of students $K \in \mathcal{K}_p$. We assume that for two programs p_1 and p_2 it holds that $p_1 = p_2 \Leftrightarrow g(p_1) = g(p_2)$. In the setting of matching internship programs with teams of students we should consider only team assignment functions g such that g assigns each student to at most one program.[3] As such, we can define *feasible* team assignment functions:

Definition 5 (*Feasible Team Assignment Functions (FTAF)*)**.** Given a set of programs P and a set of students S, a feasible team assignment function $g \in G$ is such that for each pair of programs $p_1, p_2 \in P$ with $p_1 \neq p_2$, it holds that $g(p_1) \cap g(p_2) = \emptyset$; and for all $p \in P$ it holds that $|g(p)| = m_p$.

The family of all feasible team assignments is denoted with G_{feasible}. Now we are ready to formalise our team allocation problem as follows:

Definition 6 (Team Allocation for Internship Programs Problem (TAIPP))**.** *Give a set of internship programs P, and a set of students S, the team allocation for internship programs problem is to select the team assignment function $g^* \in G$ that maximizes the overall competence proximity:*

$$g^* = \underset{g \in G_{feasible}}{\arg\max} \prod_{p \in P} \text{cp}\big(g(p), p, \eta_p^{g(p)\ *}\big) \tag{2.7}$$

The following result establishes that the TAIPP is $\mathcal{NP} - complete$ by reduction to a well-known problem in the MAS literature.

Theorem 1. *The TAIPP, with more than one program at hand, is $\mathcal{NP} - complete$.*

Proof. The problem is in $\mathcal{NP}$ since we can decide whether a given solution is feasible in polynomial time ($\mathcal{O}(\sum_{p \in P} m_p)$). We show that the problem is $\mathcal{NP} - complete$ by using a reduction from *Single Unit Auctions with XOR Constraints and Free Disposals* (referred to as BCAWDP with XOR Constraints) which is shown to be $\mathcal{NP} - complete$ [13]. In the BCAWDP with XOR Constraints, the auctioneer has N items to sell, the bidders place their bids $B_i = \langle \mathbf{b}_i, b_i \rangle$ with $\mathbf{b}_i$ be a subset of items and b_i the price. Between two bids can exist an XOR constraint–not necessarily to every pair of bids. The auctioneer allows free disposals, i.e., items can remain unsold. Given an instance of BCAWDP with XOR Constraints, we construct an instance of student-teams allocation to internship programs problem as follows: "For each item i we create a student s_i. For each program p_j of size m_{p_j} we create $\binom{|S|}{m_{p_j}}$ different bids $B_{jk} = \langle \mathbf{b}_{jk}, b_{jk} \rangle$, where $|S|$ is the number of items, $|\mathbf{b}_{jk}| = m_{p_i}$, and $b_{jk} = \text{cp}\big(\mathbf{b}_{jk}, p_j, \eta_{p_j}^{\mathbf{b}_{jk}\ *}\big)$. All bids created for program p_j are XOR-constrained bids. Moreover, each pair of bids $B_{j,k}, B_{q,l}$ such that $\mathbf{b}_{jk} \cap \mathbf{b}_{ql} \neq \emptyset$ are also XOR-constrained." Now the team allocation for internship programs problem has a feasible solution if and only if BCAWDP with XOR constraints has a solution. □

Typically, the winner determination problem for combinatorial auctions can be cast and solved as a linear program. Along the same lines, we propose how to solve the TAIPP by meas of LP in Sec 2.3. Before that, the following section characterises the search space with which an algorithm solving the TAIPP must cope.

2.2.4 Characterising the search space

The purpose of this section is to characterise the search space defined by the TAIPP. This amounts to quantifying the number of feasible team assignment functions in G_{feasible}. For that, we start by splitting the programs in P into k buckets of programs, where the programs in the same bucket require teams of the same size. That is, we have $b_1, \cdots, b_k \subseteq P$ buckets where $b_i \cap b_j = \emptyset,\ \forall i, j = 1, \cdots, k$ and $\bigcup_{i=1}^{k} b_i = P$. For each bucket b_i with $|b_i| = n_i$, it holds that $m_{p_1} = m_{p_2} = \cdots = m_{p_{n_i}} = m_i$ for all $p_1, p_2, \cdots, p_{n_i} \in b_i$; and $m_i \neq m_j$, that characterise b_i and b_j respectively, for any $i \neq j = 1, \cdots, k$. Next, we will distinguish three cases when counting the number of feasible teams in G_{feasible}:

[3] We remind the reader that these requirements are particular to our scenario of matching students to tasks. When the WeNet pilots' scenarios get finalised and their diversity dimensions identified, we will adapt our algorithm to any new requirements that might pop up.

Co-funded by the Horizon 2020 Framework Programme of the European Union

- Case I : $\sum_{p\in P} m_p = \sum_{i=1}^{k} m_i \cdot |b_i| = N$, we have exactly as many students as required by all programs in P. In this case, we seek for partition functions over P. The space of G_{feasible} is $\frac{N!}{\prod_{i=1}^{k}(m_i!)^{b_i}}$ according to Theorem 3.4.19 in [11].
- Case II : $\sum_{p\in P} m_p = \sum_{i=1}^{k} m_i \cdot |b_i| < N$, we have more students than the required ones by all programs in P. Following the Example 3.4.20 in [11], we assume one more bucket b_{k+1} containing exactly one *auxiliary* program, which requires a team of size $m_{k+1} = \sum_{i=1}^{k} m_i \cdot |b_i| - N$. Now there are $|G_{\text{feasible}}| = \frac{N!}{\prod_{i=1}^{k}(m_i!)^{b_i} \cdot \left(N - \sum_{i=1}^{k} |b_i| \cdot m_i\right)!}$ different feasible team assignment functions.
- Case III: $\sum_{p\in P} m_p = \sum_{i=1}^{k} m_i \cdot |b_i| > N$, we have less students than the required ones by all programs in P. In this case, first we need to introduce $cover(P, S) = \{P' \subset P : \sum_{p\in P'} m_p \leq N \wedge \nexists\, p' \in P - P' : m_{p'} \leq N - \sum_{p\in P'} m_p\}$ as the set that contains all the subsets of programs $P' \subset P$ such that S, P' leads to Case I or Case II, and by adding any $p \notin P'$ in p' it will lead to Case III. The number of feasible team assignment functions is:

$$|G_{\text{feasible}}| = \sum_{P' \in cover(P,S)} \frac{N!}{\prod_{i=1}^{k}(m_i!)^{b_i} \cdot \left(N - \sum_{i=1}^{k} |b_i| \cdot m_i\right)!}$$

  where variables $k, b_1, \cdots, b_k$ and $m_1, \cdots, m_k$ changes according to P'. The size of set $cover(P, S)$ depends on the total number of students, and the team sizes required by the programs in P.

Note that the number of feasible team assignment functions quickly grows with the number of programs and students, hence leading to very large search spaces.

2.3 Solving The TAIPP as a linear program

In what follows we show how to solve the TAIPP in Definition 6 as an LP. First, for each time $K \subseteq S$ and program $p \in P$, we will consider a binary decision variable x_K^p. The value of x_K^p indicates whether team K is assigned to program p or not as part of the optimal solution of the TAIPP. Then, solving the TAIPP amounts to solving the following non-linear program:

$$\max \prod_{p\in P} \prod_{k\in\mathcal{K}_p} \left(\text{cp}(K, p, \eta_p^{K\,*})\right)^{x_K^p} \tag{2.8}$$

subject to:

$$\sum_{K\subseteq S} x_K^p \cdot \mathbb{1}_{K\in\mathcal{K}_p} \leq 1 \qquad \forall p \in P \tag{2.9a}$$

$$\sum_{p\in P} \sum_{K\subseteq S} x_K^p \cdot \mathbb{1}_{s\in K} \cdot \mathbb{1}_{K\in\mathcal{K}_p} \leq 1 \qquad \forall s \in S \tag{2.9b}$$

$$x_K^p \in \{0, 1\} \qquad \forall K \subseteq S, p \in P \tag{2.9c}$$

Constraint 2.9a ensures that a program is allocated a single team. Constraint 2.9b ensures that any two teams sharing some student cannot be assigned to programs at the same time. Notice that the objective function (see Eq 2.8) is non-linear. Nevertheless, it is easy to linearise it by maximising the logarithm of $\prod_{p\in P} \prod_{k\in\mathcal{K}_p} \left(\text{cp}(K, p, \eta_p^{K\,*})\right)^{x_K^p}$. Thus, solving the non-linear program above is equivalent to solving the following binary linear program:

$$\max \sum_{p\in P} \sum_{K\in\mathcal{K}_p} x_K^p \cdot \log\left(1 + \text{cp}(K, p, \eta_p^{K\,*})\right) \tag{2.10}$$

Co-funded by the Horizon 2020 Framework Programme of the European Union

subject to: equations 2.9a, 2.9b, and 2.9c. Therefore, we can solve this LP and solve with the aid of an off-the-shelf LP solver such as, for example, CPLEX, Gurobi, or GLPK. If given sufficient time, an LP solver will return an optimal solution to the TAIPP.

At this point, it is worth mentioning that computing the objective function in 2.10 to build the LP requires the pre-computation of the values of $\mathrm{cp}\big(K, p, \eta_p^{K\,*}\big)$, which amounts to solving an optimisation problem per each pair of team and program. This is bound to lead to large linear programs as the number of students and programs grow. Furthermore, an LP solver is a general-purpose solver that does not exploit the structure of the problem. Thus, in the next section we introduce the *TAIP* algorithm, an anytime algorithm based on local search that yields approximate solutions to the TAIPP. Unlike an LP solver, TAIPP is a specialised algorithm does exploit the structure of TAIPP instances. Section 2.5 will show that TAIPP manages to outperform a general-purpose LP solver.

2.4 A heuristic algorithm for TAIPP

The TAIP algorithm consists of two stages: *(a)* finding an initial feasible allocation of students to programs, and *(b)* continuously improving the best allocation at hand by means of swaps between team members.

2.4.1 Initial team allocation

During this stage the algorithm finds an initial feasible team allocation. The algorithm sequentially picks a team for each program, starting from the 'hardest' program to the 'simplest' one. Intuitively, 'hard' programs are more selective, i.e., there are a few students that can cover it; as such, picking teams for the harder programs first is easier as we have more options (students) available. In order to evaluate the hardness of a program we will be using the notion of fuzzy entropy.

To begin with, we first evaluate the required competences from all programs, as to how hard is for the students to cover them. Looking at the competence coverage metric, we can view it as a *membership function* [14], i.e., a function that indicates in what degree a competence lies in a set of competences. Thus, fuzzy entropy [10, 1] indicates the difficulty of finding students to cover a competence. However we need to discern two extreme cases:

- all students possess competence c, i.e. $\mathrm{cvg}(c, C_s) = 1\ \forall s \in S$;
- no student can cover competence c, i.e. $\mathrm{cvg}(c, C_s) = 0\ \forall s \in S$.

Although, the above two cases result with the same fuzzy entropy (0), their intuitive interpretation is exactly the opposite. In the former case, finding a student for covering this competence within a team it is trivial since everyone can cover it. In the latter case, finding a student for covering this competence within a team it is trivial since no-one can cover it. Thus, in our definition of competence hardness we exploit the notion of fuzzy entropy, but we also embrace the intuitive interpretations above. Formally:

Definition 7 (Competence Hardness)**.** *Given a set of students S, the hardness of a competence c is defined as*

$$h(c, S) = -K \sum_{s \in S} \mathcal{H}\big(\mathrm{cvg}(c, C_s)\big) \tag{2.11}$$

where $K = 1/|S|$ is a normalization factor,

$$\mathcal{H}(x) = \begin{cases} H(x) + H(1-x) & \text{if } x \geq 0.5 \\ 4 \cdot H(0.5) - H(x) - H(1-x) & \text{otherwise} \end{cases},$$

and $H(x) = x \cdot \log(x)$.

The hardness of a competence c coincides with its fuzzy entropy when for all students the competence coverage is greater than 0.5. If for all students the competence coverage is less than 0.5 the competence hardness is the constant $4 \cdot 0.5 \cdot \log(0.5)$ minus the fuzzy entropy. The constant $4 \cdot 0.5 \cdot \log(0.5)$ derives from the fuzzy entropy of point 0.5: coverage 0.5 indicates that all students are neither good nor bad for this competence, as such hardness in point 0.5 shall be the median, thus the maximum of the competence hardness is $2 \cdot \big(0.5 \log(0.5) + (1-0.5) \log(1-0.5)\big) = 2 \cdot 2 \cdot 0.5 \log(0.5)$. Graphically, the competence hardness is shown in Fig 2.1.

Co-funded by the Horizon 2020 Framework Programme of the European Union

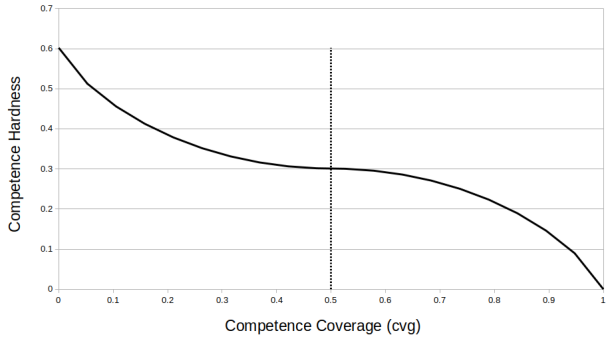


Figure 2.1: Competence hardness.

The degree of hardness of a program is determined by the available set of students' difficulty for covering the program's competencies. We remind the reader that each required competence is accompanied by an importance weight (Sec 2.2.1). Thus, consider a program where its most important competence $c_{important}$ (i.e., the competence with the highest $w_p(c)$) is very difficult to be covered $h(c_{important}, S) \simeq 4 \cdot H(0.5)$, then this program is extremely hard. On the other hand, if a specific competence c is somewhat difficult to be covered ($h(c_{important}, S) \to 4 \cdot H(0.5)$), but it is not very important ($w_p(c) \to 0$), then the program is not that hard.

Definition 8 (Program Hardness)**.** *Given a set of students, the hardness of a program p is defined as the aggregation of the hardness for the competences in the program weighted by the importance of each competence: $h(p, S) = W \cdot \sum_{c \in C_p} \frac{1}{h(c,S)+\epsilon} \cdot w_p(c)$, where $W = \frac{1}{\sum_{c \in C_p} w_p(c)}$ is a normalization factor, and ϵ is a small positive constant.*

In words, the more important and the more difficult a competence is to be covered, the more difficult is to find students with high competence proximity for the program, consequently the harder the program is considered to be. Note that both $w_p(c)$ and $h(c)$ are non-negative, so the hardness of one competence cannot be counteracted by the non-hardness of another within the same program.

Algorithm 1: Initial Team Allocation

input : Students S, programs P, (optionally) sorting order $order$ for P
output : team assignment function g

$V_p \leftarrow \bigcup_{p \in P} C_p$;
for $c \in V_p$ **do** $\mathrm{hc}[c] \leftarrow h(c, S)$;
for $p \in P$ **do** $\mathrm{hp}[p] \leftarrow h(p, S)$;
sort P in descending order *wrt* hp;
while $P \neq \emptyset$ **do**
  $p \leftarrow$ pop first from P;
  if $|S| \geq m_p$ *and* $\mathrm{hp}[p] < 1$ **:**
    sort S maximizing coverage in benches of $|C_p|$;
    `/* assign team to program */`
    $g(p) \leftarrow m_p$ first students in S;
    $S \leftarrow S \setminus g(p)$;
    update values in hc;
    if $|S| < 1$ **:** break;
return g;

Co-funded by the Horizon 2020 Framework Programme of the European Union

2.4.2 Improving team allocation

In the second stage we perform a number of random 'movements', until convergence to a local or global maximum. The second stage starts with the team assignment produced in the first stage. Thereafter, we iteratively improve the current team assignment either *(i)* by employing crossovers of students between two programs, and/or *(ii)* by swapping assigned students with available ones if they exist. Specifically, following Algorithm 2 within an iteration we randomly pick two programs (line 4) and attempt to improve the competence proximity of the pair by exhaustively searching of all possible crossovers of the students assigned to these programs (line 7). However, in order not to computationally overload our algorithm with repetitive exhaustive searches, we perform it only if the following two conditions hold:

1. the two programs share similar competencies; and
2. some student in one of the teams improves the coverage of some competence of the other team.

In order to evaluate if two programs share similar competencies we exploit the *Hausdorff* distance [6]. The Hausdorff distance between the required competencies of two programs p_k and p_l is defined as:

$$\text{dist}(C_{p_1}, C_{p_2}) = \max\{\min_{c \in C_{p_1}}\{\text{cvg}(c, C_{p_2})\}, \min_{c \in C_{p_2}}\{\text{cvg}(c, C_{p_1})\}\}.$$

The above two conditions encode the potentiality of finding an improvement for these two programs, and whether it is worth performimg an exhaustive search. In the exhaustive search, given the students $g(p_k) \cup g(p_l)$ we produce all possible partitions that contain two teams of sizes m_{p_k} and m_{p_l}. For each of these partitions we compute the competence proximity of the pair of programs, and yield with the optimum one, i.e., with the partition that achieves the greater competence proximity.

In case we did not achieved any improvements from the previous step and there are available students, i.e. student that have not been assigned to any program, we attempt to swap assigned students with available ones (line 9). That is, we randomly pick a student from either of the two programs, and try to randomly swap them with a student in $S_{available}$. If we achieve an improvement we keep this alteration, otherwise we repeat this process for a fixed number of attempts. In case we succeeded to imporove the competence proximity of the pair, we update the team assignement g, the set of availble students $S_{available}$, and the current overall competence proximity (lines 10-15).

In order to overcome the possibility of a series of unsuccessful attempts between random programs, we force a more 'systematic' search, which we call local search, on the programs. This local search is performed after a constant number of iterations (line 16). In the local search (line 17) we go through *all* programs in P, swap all members, and check whether some swap improves the overall competence proximity–in the swaps we consider all students: both assigned and available.

Note that Algorithm 2 is anytime algorithm that can yield a result after any number of iterations indicated by the user. However, in its generality, we adopt a notion of convergence in order to terminate the algorithm. That is, we terminate the algorithm (line 3):

- after a number of iterations without no improvements; *or*
- if we reach an overall competence proximity close to 1.

Note that we added the latter termination condition in order to avoid unnecessary iterations until convergence, due to the fact that the maximum value the overall competence proximity can reach is 1. We remind the reader that the competence proximity is the Nash product of the individual competence proximity of the teams to their assigned program (Eq 2.8), and each individual competence proximity lies in $[0, 1]$ (Def 4). However, we should make clear that the overall competence proximity does *not always* reach 1, but that it can *never exceed* 1.

2.5 Empirical analysis

The purpose of this section is to empirically evaluate the TAIP algorithm along four directions:

- the quality of the solutions that it produces in terms of optimality;

Co-funded by the Horizon 2020 Framework Programme of the European Union

Algorithm 2: Improve Team Allocation

input : Students S, programs P, team assignment g
output : improved team assignment g

$S_{\text{available}} = S \setminus \bigcup_{p \in P} g(p)$;
$\text{current\_cp} = \prod_{p \in P} \text{cp}\big(g(p), p\big)$;
while *non_improved* **and** $1 - \text{current\_cp} > \varepsilon$ **do**
  $p_k, p_l \leftarrow$ randomly select two programs from P;
  $\text{pair\_cp} = \text{cp}\big(g(p_k), p_k\big) \cdot \text{cp}\big(g(p_l), p_l\big)$;
  if potentiality(p_1, p_2, g) **:**
    $\text{new\_cp}, K_k, K_l \leftarrow$ exhaustiveSearch(p_1, p_2, g);
  else :
    $\text{new\_cp}, K_k, K_l \leftarrow$ localSwaps$(p_1, p_2, g, S_{\text{available}})$;
  if $\text{new\_cp} > \text{pair\_cp}$ **:**
    $g(p_k) \leftarrow K_k$;
    $g(p_l) \leftarrow K_l$;
    $S_{\text{available}} \leftarrow S \setminus \bigcup_{p \in P} g(p)$;
    $\text{current\_cp} \leftarrow \text{current\_cp} \cdot \frac{\text{new\_cp}}{\text{pair\_cp}}$;
    $\text{pair\_cp} \leftarrow \text{new\_cp}$;
  if *time for local search* **:**
    $g, S_{\text{available}}, \text{current\_cp} \leftarrow$ localSearch$(P, g, S_{\text{available}})$;
return g;

- the quality of the solutions produced by the initial stage;
- the time required by TAIP to produce optimal solutions with respect to CPLEX, an off-the-shelf linear programming solver; and
- the time required by TAIP to yield optimal solutions as the number of students and programs grow.

Overall, our results indicate that TAIP significantly outperforms CPLEX, and hence it is the algorithm of choice to solve the Team Allocation for Internship Programs Problem introduced in this chapter. Next, in Sec 2.5.1 we describe the settings employed in our experiments, whereas Sec 2.5.2 dissects our results.

2.5.1 Empirical settings

For our experimental evaluation, we were looking for ways to define competences in universities, and as such, we used an existing competence ontology provided by *Fondazione Bruno Kessler* (https://www.fbk.eu/en/); and generated synthetic data in the following way:

Internship program generation For each program p

1. select the required team size $m_p \sim \mathcal{U}\{1, 3\}$
2. select the number of required competences $|C_p| \sim \mathcal{U}\{2, 5\}$
3. randomly choose $|C_p|$ competences from the ontology
4. the required level function is set to $l_p(c) = 1,\ \forall\, c \in C_p$
5. the weight function is $w_p(c) = \mathcal{N}\big(\mu = \mathcal{U}(0, 1), \sigma = \mathcal{U}(0.01, 0.1)\big)$ bounded in $(0, 1]$ for all $c \in C_p$.

Co-funded by the Horizon 2020 Framework Programme of the European Union

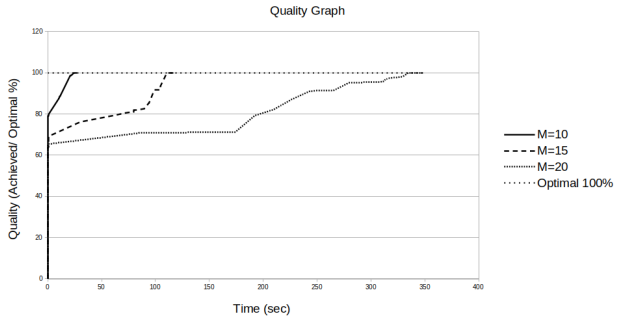


Figure 2.2: Solution quality achieved by TAIP along time.

Student generation For each program p

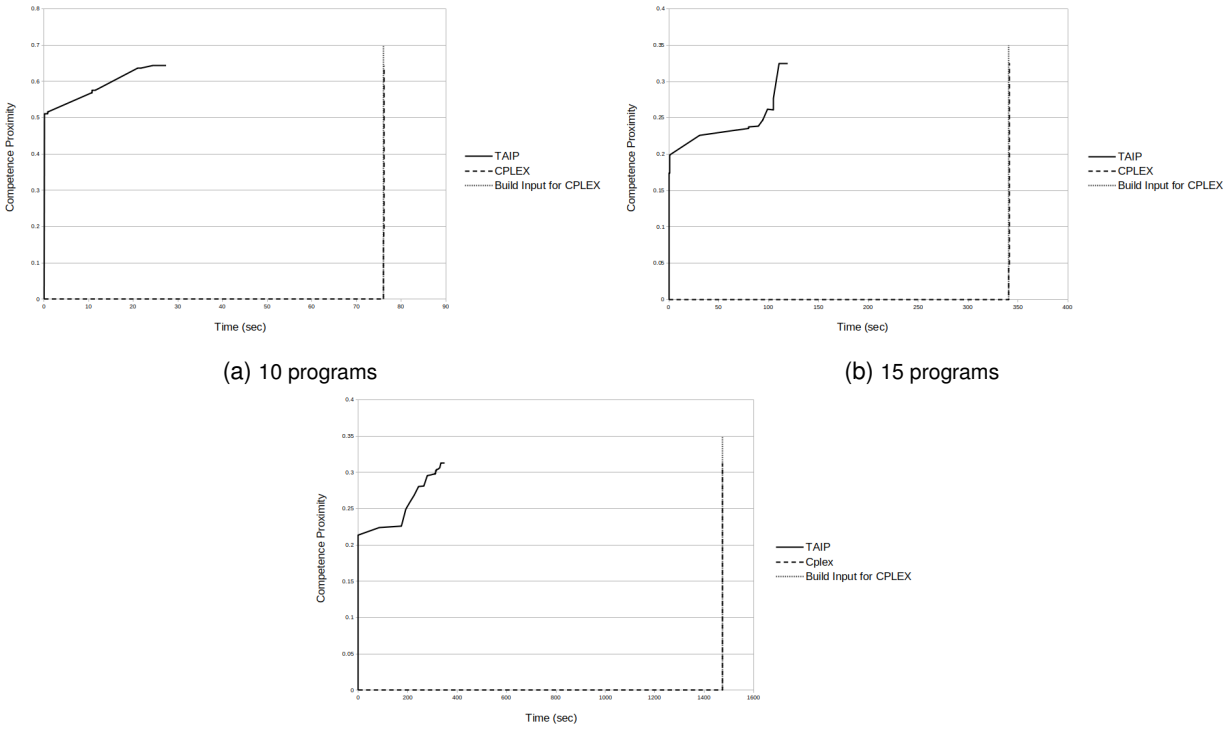
1. generate m_p new students such that for each student s: there are competences $c \in C_p$ and $c' \in C_s$ such that c' is *(i)* identical to c; or *(ii)* a child-node of c in the ontology; uniformly selected among the options.

With these generators we constructed 60 different TAIPP instances, which are shown in Table 2.2. We solve each problem instance with both TAIP and the IBM CPlex linear programming (LP) solver. The experiments were performed on a PC with Intel Core i7 (8th Gen) CPU, 8 cores, and 8Gib RAM. Moreover, we employed IBM ILOG CPLEX V12.10.0. For all implementations we used Python3.7.

2.5.2 Results

Quality analysis. Using the optimal solutions yielded by CPLEX as an anchor, we can evaluate the quality of the solutions computed by the TAIP algorithm. Notice that for all problem instances, TAIP reaches the optimal solution. More precisely, for every problem instance, TAIP achieved a solution whose value, in terms of competence proximity, is the same as the value of the optimal solution computed by CPLEX. Fig 2.2 shows the average *quality ratio* of TAIPP with respect to CPLEX along time for the problem instances in Table 2.2. We calculate the quality ratio by dividing the competence proximity computed by TAIP by the optimal value computed by CPLEX, and it is depicted as a percentage (%).
Runtime analysis. The greatest advantage of TAIP is that it is way much faster than CPLEX. As shown in Fig 2.3 TAIP reaches optimality in less than half of the time required by CPLEX. Specifically, for problem instances with 10 programs, TAIP requires on average $\sim 40\%$ of the time CPLEX needs, i.e., is $\sim 60\%$ faster. As to problem instances with 15 programs, TAIP requires on average $\sim 45\%$ of the time employed by CPLEX ($\sim 55\%$ faster). Finally, for problem instances with 20 programs, TAIP requires on average $\sim 29\%$ of the time spent by CPLEX ($\sim 71\%$ faster). Therefore, the larger the size of the problem instances, the larger the benefits for TAIP with respect to CPLEX. Here we should note that the time consuming task for CPLEX is the building of the LP encoding the problem, while solving the actual problem is done in seconds. This indicates that the problem instances under investigation are rather large than hard: as the number of programs increases, so does the number of students, resulting in large linear programs.
Anytime analysis. Last but not least we present our results on the anytime behavior of TAIP, as shown in Fig 2.4. We observe that after completing the initial stage described in Sec 2.4.1, the solution quality produced by TAIP reaches 80%, 70%, and 65% of the optimal solution, for problem instances with 10,15 and 20 programs respectively. Furthermore, TAIP reaches quality 80% in $0.001 \times t_{CPLEX}$ for 10 programs, 70% in $0.025 \times t_{CPLEX}$ for 15 programs, and 65% in $0.0002 \times t_{CPLEX}$ for 20 programs, where t_{CPLEX} is the time CPLEX needs to compute the optimal solution. Moreover, in all investigated settings we reached 80% quality in less than 20% of the time CPLEX needs: 0.1%, 20%, and 13.5% of CPLEX time for 10,15, and 20 programs.

Co-funded by the Horizon 2020
Framework Programme of the European Union

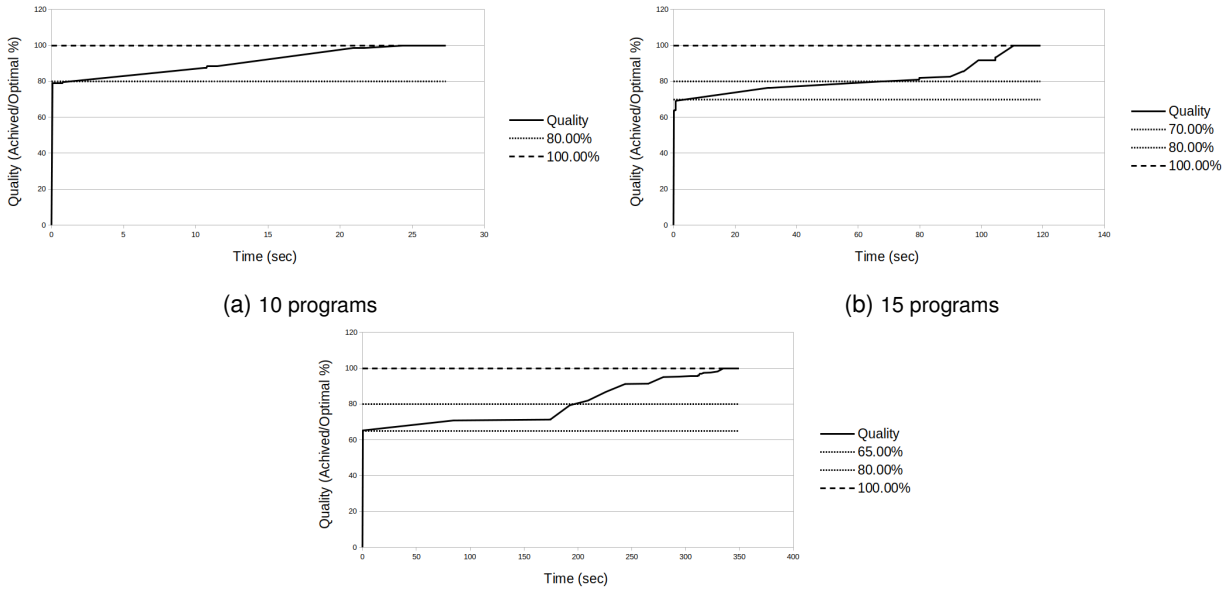


(a) 10 programs

(b) 15 programs

(c) 20 programs

Figure 2.3: Average Competence Proximity vs Time



(a) 10 programs

(b) 15 programs

(c) 20 programs

Figure 2.4: Anytime Behavior

Co-funded by the Horizon 2020
Framework Programme of the European Union

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Average
N=#Students	18	20	21	19	22	19	24	18	19	20	23	20	18	19	25	21	25	20	17	13	20.5

(a) Family of datasets with 10 programs

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Average
N=#Students	23	33	32	29	33	40	31	32	28	25	27	31	29	28	32	29	29	32	34	29	30.6

(b) Family of datasets with 15 programs

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Average
N=#Students	44	45	44	45	38	42	42	47	41	44	37	36	42	37	47	32	40	37	44	43	41.35

(c) Family of datasets with 20 programs

Table 2.2: Synthetic problem instances.

2.6 Conclusions and future work

As illustrated in the opening of this chapter, how to group people cannot be conducted in a generic way, independent of the goal of the team. As such, any team formation algorithm is influenced by the expected goal of the team. Currently, the WeNet pilot scenarios are still being designed, the diversity dimensions are being laid out, and the exact goals of teams are yet to be finalised. As soon as these details are agreed upon, the plan will be to revisit our proposed algorithm to take into consideration the goal of the agreed upon pilot scenarios and the possible diversity dimensions that are relevant for team formation.

Nevertheless, for now, this chapter has formally defined the problem of allocating teams to tasks. We first studied the problem's complexity and characterised its search space.Thereafter, we provided an encoding to optimally solve the TAIPP by means of linear programming. Then, we proposed a novel, heuristic anytime algorithm, TAIP. Finally, we conducted a systematic comparison of TAIP versus the CPLEX LP solver when solving TAIPP problem instances. Our experimental evaluation showed that TAIP outperforms CPLEX in time, mainly because of the extremely large input that the latter requires. Moreover, TAIP always managed to reach the optimal solution for the problem instances under investigation. Specifically TAIP converged to the optimal in less than 40% of the time required by CPLEX, and achieved a quality of 80% in less than 20% of the time required by CPLEX.

Co-funded by the Horizon 2020
Framework Programme of the European Union

Bibliography

[1] S. Al-sharhan, F. Karray, W. Gueaieb, and O. Basir. Fuzzy entropy: a brief survey. In *10th IEEE International Conference on Fuzzy Systems. (Cat. No.01CH37297)*, volume 3, pages 1135–1139 vol.2, Dec 2001.

[2] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. Online team formation in social networks. In *WWW'12 - Proceedings of the 21st Annual Conference on World Wide Web*, pages 839–848, 04 2012.

[3] Ewa Andrejczuk. Artificial intelligence methods to support people management in organisations. Doctoral, 05/2018 2018.

[4] Ewa Andrejczuk, Rita Berger, Juan A. Rodríguez-Aguilar, Carles Sierra, and Víctor Marín-Puchades. The composition and formation of effective teams: computer science meets organizational psychology. *Knowledge Eng. Review*, 33:e17, 2018.

[5] Ewa Andrejczuk, Filippo Bistaffa, Christian Blum, Juan A. Rodríguez-Aguilar, and Carles Sierra. Synergistic team composition: A computational approach to foster diversity in teams. *Knowledge-Based Systems*, 182(104799), 10/2019 2019.

[6] AURA Conci and CS Kubrusly. Distance between sets-a survey. *arXiv preprint arXiv:1808.02574*, 2018.

[7] Chad Crawford, Zenefa Rahaman, and Sandip Sen. Evaluating the efficiency of robust team formation algorithms. In Nardine Osman and Carles Sierra, editors, *Autonomous Agents and Multiagent Systems*, pages 14–29, Cham, 2016. Springer International Publishing.

[8] Mehdi Kargar, Aijun An, and Morteza Zihayat. Efficient bi-objective team formation in social networks. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 483–498, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] Y. Li, Z. A. Bandar, and D. Mclean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):871–882, July 2003.

[10] Aldo De Luca and Settimo Termini. Entropy of l-fuzzy sets. *Information and Control*, 24(1):55 – 73, 1974.

[11] R. B. Maddox. *Mathematical Thinking and Writing: A transition to abstract mathematics*. Academic Pr, San Diego, 1st edition, 2002.

[12] N. Osman, C. Sierra, F. Mcneill, J. Pane, and J. Debenham. Trust and matching algorithms for selecting suitable agents. *ACM Trans. Intell. Syst. Technol.*, 5(1):16:1–16:39, January 2014.

[13] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Winner determination in combinatorial auction generalizations. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, page 69–76, New York, NY, USA, 2002. Association for Computing Machinery.

[14] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

Co-funded by the Horizon 2020 Framework Programme of the European Union